Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document No. 245

A Network Unix System. Vol. 3: User Telnet Implementation for Platform

by
J. S. Goldberg

April 1978

CAC Document Number 245

A Network Unix System:

Volume 3

User Telnet Implementation for Platform

by

J. S. Goldberg

Prepared for the
Department of Defense

Center for Advanced Computation and
Computing Services Office of the
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

April 1978

Approved for release: Karl C. Kelley

Karl C. Kelley, Principal Investigator

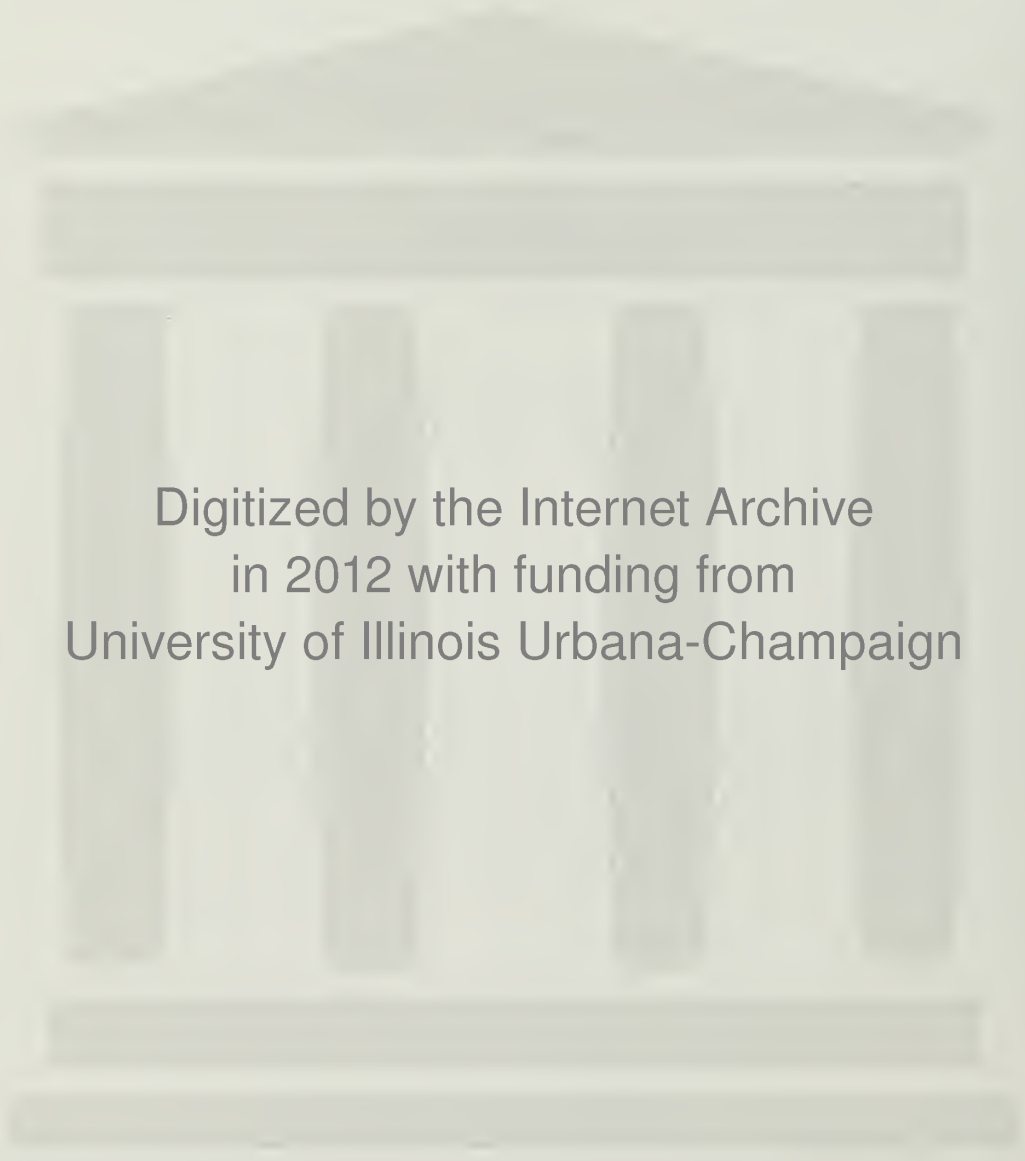

Table of Contents

Table of Contents (cont'd)

1 Overview and Introduction

One of the uses for a network such as the Arpanet or Platform network is allowing the user to access any host computer on the network for timesharing. Work on the local system need not involve the network at all, but to get across the network to a foreign computer system one needs a way to permit network access. For the local timesharing user a program to allow access to the net is sufficient; for the foreign system, a program to allow the remote user access to the facilities is required.

The entire scheme for cross-net timesharing use is called Telnet. Telnet is divided into the two parts described above: User telnet, allowing the local user to access other portions of the network; Server telnet, which permits a user access to a machine which is not local to him/her. The two telnets (User in the user's local machine and Server in the remote machine the user wishes to use) provide a framework to allow easy access to machines across the net. Within this framework are provisions to transmit special function commands to telnet for more precise control of the connection and remote process(es).

1.1 The Network Virtual Terminal

Both telnets initially assume the terminal characteristics of the Network Virtual Terminal (NVT). This is basically a teletype model 37 (upper/lower case, half duplex, no frills) - in other words, a really bare-bones terminal. This rather rudimentary terminal was chosen to be the standard to avoid penalizing users with modest terminal equipment. In order to avoid penalizing those users with elegant terminal equipment, the NVT may be extended to include various fancy features, provided that both telnets agree on the desired extensions. Some of the aforementioned special function commands are used to communicate desire to change terminal characteristics. This process is known as 'option negotiation' and will be dealt with later in detail.

1.2 General Operation of Telnet

The user telnet program may be logically divided into several sections to perform various functions. The major functions are: Get characters from the terminal and transmit them to the network; interpret certain of these characters (or strings of them) as commands to user telnet; receives characters from the network and transmit them to the user's terminal. Since the purpose of telnet is shuffling characters back and forth, it must be made as convenient as possible to do so. Toward this end, several different terminal modes are provided which dictate certain of the parameters for handling characters. Further, a shortcut method (character mapping) is provided, along with methods for directing the control of the state of user telnet,

and the network connection in general. Naturally, with all these functions, we will absolutely need at least one more - help capabilities may be invoked for any of the telnet command keywords.

This document describes Unix Telnet. Since the telnet user needs not interact with server telnet, little if any information on it appears in the user's guide (but it is dealt with in the maintenence section). Program operation and function are discussed in several sections, which get progressively more detailed.

2 User Telnet User's Guide

This section of the document refers to necessary information for operating user telnet. Included are instructions for running user telnet, descriptions and effects of the various commands user telnet provides, and some of the conceptual highlights of telnet. Much of the description of telnet's capabilities is found in the command descriptions, with further explanation in the elaboration section - all telnet users are urged to read these sections carefully, and refer to them for additional information after using telnet for a while and 'getting a feel' for the program.

2.1 Simple use of User Telnet

Actually, user telnet is quite easy to use. It has been said that for most reasonable hosts one need never use telnet commands other than those for establishing and closing a connection. The relatively large number of commands need not frighten the novice or undemanding user; they are provided for the experienced user who wishes telnet to do clever manipulations.

2.1.1 Invoking User Telnet

Running user telnet is quite easy; all one needs to type is:

telnet

whereupon, telnet will respond with a herald message and give you a prompt (currently '^'[1] by default). The appearance of the prompt indicates that telnet is ready to accept a command.

Of course, if you don't know what to do at a command prompt (or don't need to use any commands), you can circumvent seeing one by typing:

telnet <hostname>

where <hostname>[2] represents the name of a network server host.

---

[1] The character '^' appears as a carat on some printers, and an up-arrow on others.

[2] The lexical construction '<word>' is used to indicate one of many possible parameters. The angle brackets ("<", ">") are not included. The item enclosed in the brackets is mnemonic for the parameter, but is not the parameter itself.

Telnet then attempts to connect you to the specified host. For mnemonic ease, you may instead type:

con <hostname>

(in fact, 'con' is a synonym for 'telnet') .[3] 'Con' is short for 'connect'. For example, to initiate a connection to a foreign host named overthere, one would type:

con overthere

Of course, you could use 'telnet' instead of 'con'.

2.1.2 Extra Connection Parameters

For those who want a bit more control over their connection requests, parameters are available, thus:

telnet [hostname] [<parameters>]
Or...
con [hostname] [<parameters>]

The optional[4] parameters to telnet, specified above, consist of the flag character '-', a switch key-letter, and additional argument(s) which are usually integer(s). Telnet will interpret any integer as decimal, unless the first digit is a '0' - in which case it will be interpreted as octal. Switches and their arguments (with integers represented as '#') are:

-a # desired message allocation size

-d # # specifies a direct connect (as opposed to normal ipc) foreign socket. a 36 bit quantity which is specified as two 16 bit integers, high order portion first. usually, one wants the s switch, not this one.

-h # host number of the desired host (use without the

---

[3] A word about a possible confusion is in order here. Both 'con' and 'telnet' will get you into telnet, but neither is a telnet command - they are both names for the user telnet program. Later, we shall see that there are also telnet commands by these names; remember that they are not the same thing.

[4] Throughout this document, optional items will be enclosed in square brackets ("[", "]"). Hopefully, this will not cause confusion with footnote references.

hostname)

-d # # direct connect foreign socket (32 bit number, with the high order half first).

-s # foreign socket to connect to (default is 1)

-t # number of seconds before timeout (default 120)

These switches are all optional and may be in any order; however, the hostname (if present) must come first. Since the <hostname> and "-h" parameters both specify the foreign host, only one should be used for any given connection. If you forget to include a number, a zero value will be assumed and could lead you to unexpected results. Refer to the connection discussion in section 2.3.1 for more detail.

2.2 Telnet Commands

In order to make telnet versatile, a wealth of commands is available. The intention is to satisfy the demanding user as well as those who have no need for telnet's full power. Telnet distinguishes a command line (that is, a line it is to interpret) from normal data to go to the network by the presence of a special escape character as the first character on the line (see the escape command). Obviously, if there is no network connection open, everything you type must be intended for telnet's interpretation, and the escape character is optional.

When choosing the names for the telnet commands, efforts were made to keep them distinct from the names of Unix commands. This consideration permits the user even greater flexibility in using telnet because if telnet does not recognize the command entered, it trys to execute it via a shell. In other words, if you type a string to telnet that it does not know about, it assumes that command to be a Unix command, and attempts to execute it, complete with whatever arguments you've typed.

Many telnet commands may be used in different contexts. Some commands may be used by themselves, some are prefixes to other commands, some may be optionally prefixed, and some require a prefix. The prefixing rules are designed to avoid confusion for the novice in learning telnet commands, and to prevent accidental errors. Because most commands may be used with or without prefixes, the distinction between commands and their parameters sometimes becomes blurred. In some cases, keywords will be spoken about - these are the character strings that telnet recognizes which have significance, be they commands, prefixes, etc.

2.2.3 Command Classes

Telnet's commands can be divided into two classes: those related to control of the communication path to a foreign host; and those which control the state of the telnet process and allow the user to access various help functions. Commands in the first set deal with the opening of connections to foreign network servers, conversations with them, and the termination of those connections. These connections follow the telnet protocol[5] The second command class controls local functions and parameters such as: the treatment of characters obtained from the standard input file, help functions, status inquiries, and the state of the local terminal.

In the following sections the commands are split into these two functional groups and alphabetized within each group to ease later reference. In the case of all keywords, the full string need not be given - merely enough to be unique.

2.2.4 Lexical Conventions

Several shorthand notations are used in the command descriptions, especially in the line giving the valid syntax for each command. These are:

a) Optional items are enclosed in square brackets ("[", "]")
b) Command syntax lines list the entire command - prefixes, parameters, and all.
c) Parameters are usually specified by mnemonics enclosed in angle brackets ("<", ">")
d) Unimplemented commands are enclosed in double curly-braces ("{{", "}}")

2.2.5 Connection related commands

[send] ao

Causes immediate transmission of a telnet Abort Output command to the serving host.

[send] ayt

Causes immediate transmission of a telnet Are You There command to the serving host.

---

[5] At this time, the Platform protocol specifications are extremely similar to the specifications for the Arpa network telnet protocol, which may be found in the Arpa Telnet Protocol Specification (NIC 18639).

[send] brk
Causes immediate transmission of a telnet BReaK command to the serving host.

bye
Synonym for end; closes a connection, if open, and exits telnet.

close
Closes any open connection and returns to telnet command level.

connect [hostname] [<parameters>]
Attempts to open the requested connection to a foreign host. This command is a synonym for the 'telnet' command.

[send] del
Causes immediate transmission of an ASCII DELete character (octal 0177) to the serving host.

disconnect
This command closes the current net connection; it's a synonym for close.

send ec
Causes immediate transmission of a telnet Erase Character command to the serving host. (To avoid confusion with the 'echo' command, the prefix is required).

send el
Causes immediate transmission of a telnet Erase Line command to the serving host. (To prevent confusion with the 'eline' command, the prefix is required).

end
Exits from telnet, after closing any open connection and restoring things to normal. This command is a synonym for the 'bye' command.

exit
Synonym for end; closes a connection, if open, and exits telnet.

[send] goa
Causes immediate transmission of a telnet GO-Ahead command to the serving host.

[send] ip

Causes immediate transmission of a telnet Interrupt Process command to the serving host.

[send] nop
Causes immediate transmission of a telnet NoOPeration command to the serving host.

open [hostname] [<parameters>]
Synonym for the telnet command; see below.

{{ option <opt> <mode> }}
Prefix for the telnet options. Tells telnet to negotiate with the remote server to set telnet option <opt> to the specified mode (either 'on' or 'off'). See section on telnet protocols and options.

quit
Synonym for end; closes connection (if open) and exits telnet.

send <params>
Provided to allow the user to send unusual things down the open connection. In addition to prefixing many commands, send may also:
a) Send control characters. By putting a '/' in front of a character, the top three bits will be zeroed before the character is sent.
b) Send arbitrary bytes, numerically specified. If a digit string is encountered, digits are collected until the first non-digit; the least significant byte of the result of the conversion is then sent. If the first digit is a '0' an octal radix is used, otherwise, decimal is assumed. If, in the process of converting the digits into binary, a conversion error occurs (i.e. no digits at all, or non-octal digit in a number supposed to be octal, thus: 0281) a zero is assumed.
There may be an arbitrary number of parameters to send on the command line - parameters are separated by space, tab, or comma.

[send] synch
Causes immediate transmission of a telnet SYNCH sequence to the serving host.

telnet [hostname] [<parameters>]
Attempts to open the requested connection to a foreign host. Note that the Unix command 'telnet' and the telnet command 'telnet' (and its synonyms)

are syntactically identical; however, to open a connection, a host must be specified.

Most switches take additional argument(s) which are usually integer(s). Telnet will interpret any integer as decimal, unless the first digit is a '0' - in which case it will be interpreted as octal. Switches and their arguments (with integers represented as '#') are:

-a # desired message allocation size

-d # # specifies a direct connect (as opposed to normal ipc) foreign socket. a 36 bit quantity which is specified as two 16 bit integers, high order portion first. usually, one wants the s switch, not this one.

-h # host number of the desired host (use without the hostname)

-s # foreign socket to connect to (default is 1)

-t # number of seconds before timeout (default 120)

2.2.6 Local telnet control and help function commands

[set] charmode

Charmode forces telnet to send each character typed to the net immediately (exception: see the 'escape' command). See the elaboration of connection modes. (Section 2.3.2)

commands

Types a list of all the command keywords known to telnet.

{{[set] display <c> "string" }}

Establishes a correspondence between a character <c> received from the net and an arbitrary, user-defined character string. Maps the received character into the specified string for typing on the terminal.

[set] echo <mode>

Controls local echoing. The mode setting is either 'on' or 'off'; initially, echo is turned on.

Note: setting the local echo takes place immediately and remains in effect until countermanded.

[set] eline [<c>]

If the optional argument is present, the endline character is set to the first character of the argument. Whether or not the argument was present the now-current setting of the endline character is typed. The endline character is the character upon which a line of text is sent to the serving host. Note: this function is only useful in message mode (see below), as each character is sent immediately in charmode, and the endline character is system-defined in linemode. Default endline character is system-defined (but usually newline or carriage return).

[set] escape [<c>]

If the optional argument is present, the escape character is set to the first character of the argument. Regardless of the presence or absence of the argument, the current setting of the escape character is then typed. If the first character on a line is the escape character while a connection is open, the rest of the line is interpreted to be a telnet command. The default escape character is "^".

help [<keywords>]

Invokes the help function to tell you some useful things about the keyword(s) specified. If a keyword is unrecognized, or no keyword is specified, instructions for using the help command are given.

hosts

Produces, on the terminal, a brief list of all known network hostnames. If the host is up (to the best of the NCP's knowledge), the name is followed by a "^" character. (This command actually runs the hosts program).

{{[set] key <c> "<string>" }}

NOTE: this feature is not yet implemented, but if the description interests you, see where the 'load' command leads you.

Establishes a mapping between a certain keyboard character <c> and an arbitrary string. When key <c> is typed, the user-defined string will be sent to the server, not the actual key.

[set] linemode

Places telnet in the default Unix line-at-a-time mode. Anything you type is subject to the Unix system editing characters and conventions (i.e. the endline, backspace, and line delete characters are system defined). Further, telnet itself does not get any characters you type until you type the system endline (this is often the explanation for unusual-looking things happening). The telnet settable-endline character has virtually no effect in this mode. See also the 'eline' command. This is the default mode during connections, and the only mode available while no connection is open. For further discussion, see the elaboration of connection modes.

load [<pathname>]

The load command provides a limited character mapping facility. Load causes a character map set to be loaded into a telnet internal table. Analogous to the function of the set key command, load provides for mapping specified typed keys into arbitrary strings - the key is not sent to the server, but when it is typed, the string corresponding to it is sent. The pathname is the name of a character mapping-set file produced by the 'mkcharmap' program (refer to it's manual page for further details). If a pathname is present, telnet will try to load that file (if the pathname is not a full pathname - e.g. does not begin with '/' - the user's home directory is used as a base for the pathname). When the pathname is absent, telnet will attempt to load the file <home directory>/character_set.

[set] msgmode

Puts telnet into message mode. In this mode, telnet waits for the settable-endline character (see 'endline' command) before sending any characters to the server. In other words, it allows one to send line-at-a-time messages to the server and choose the endline character too. For further discussion, see the elaboration of connection modes.

[echo] off

Turns local echoing off. See also the 'echo' command.

[echo] on

Turns local echoing on. See also the 'echo' command.

set <cmd> <args>
This command is a prefix to many other commands (i.e. 'charmode', 'escape') and is provided for naming clarity. Usually it is not necessary, but is helpful to the novice for the sake of mnemonic commands.

settty
This command does nothing to the terminal, as the name implies - it types out telnet terminal mode information and is used soley for debugging telnet.

status [<hostname>]
To inquire about a host's status one uses this command. If the hostname is absent, the status of the IMP/interface hardware is given.

[set] tenex
Purely a convenience command, this puts the terminal into charmode with local echoing off - the preferred mode for TENEX connections. See the 'charmode' and 'echo' commands, and the elaboration on connections for more detail.

wait <nsecs>
Causes telnet to suspend keyboard processing for the specified number of seconds. (<nsecs> is an integer). Useful for logging into a remote host via a character mapping trick (see the manual page for the 'mkcharmap' program).

2.3 Elaboration on Some Concepts

This section provides a semi-detailed discussion of some of the concepts involved in operating user telnet. These discussions are intended for the novice to skim, and then reread at a later date, after having a bit of experience with telnet. Not all of the information here will be useful initially, but the general ideas are useful for deciding, for example, the terminal mode you wish to operate in. Particularly recommended for the novice are the discussions of connections and terminal modes, with protocols running a very close second.

2.3.1 Connections

When you tell user telnet to connect you to a foreign host, and the connections goes thru, you probably don't even consider how it works. Here will be a much simplified explanation of the mechanism of connecting to another host on the net. Actually, the matter is conceptually quite simple: telnet requests that the

nal, and kill the user telnet process because it's managed to get itself 'hung'. This is most certainly NOT the case. You have connected to someplace that does not have a server listening and so absolutely nothing will happen on the connection. This is the main hazard of the '-s' parameter.

However, the '-s' switch has several good uses. The normal user would not need to connect to a special socket for testing a new version of a server, but may wish to use another facility of a remote host. For example, the well known socket for FTP (the File Transfer Program) is socket three. One could use telnet to connect to a server FTP in order to send network mail, if the user's site lacks a specific network mailer. Some sites may have additional special programs or experimental services running on advertized sockets that one may wish to access.

2.3.2 Local terminal and connection modes

There are several modes available for communication with a foreign host once you've opened a connection. Each mode deals with when characters are transmitted from your local computer to the serving host. Further, you have control over some of the local modes that your terminal can be set into. These two functions are quite orthogonal, so we'll take the easiest first.

The only local terminal mode you really have direct control of is the echoing function. That is, you can tell telnet to be sure that anything you type is, or is not, echoed back to your terminal by the local system. If your terminal is half-duplex, it does all the echoing within itself and the system never needs to echo anything. However, if it's full duplex, there are some times that the characters you type should not get back to the terminal to appear on it. For example, when you login you really don't want your password echoed lest someone else discover it. If you connect to a foreign host that will not echo your password half the battle is won; now all you have to worry about is the local telnet. Unless you've arranged for the remote host to do all your echoing (as 'tenex' mode does) or the foreign server is smart enough to tell user telnet to turn off echo at the proper time, you will type your password to telnet, who doesn't know that it's a password, and it will appear in front of you for all to see. For this reason, the 'set echo on' and 'set echo off' commands are provided. Issuing one will immediately set the local echo as requested (they have no effect over what the foreign host does), and it'll stay that way until you reset it.

Talking to the foreign host can be done various ways. You can have the system save the characters you type until you finish the line (by typing the system-defined end-line character (usually a newline or carriage return)), at which point the system gives the line to telnet for transmission to the foreign host

(linemode). The other extreme is to have the system give telnet every character you type immediately so telnet can transmit characters to the network as soon as you type them (charmode). A compromise is also available: have the system give each character to telnet immediately, but have telnet save up whole lines and transmit them when you type a self-defined end-line character (msgmode).

Each scheme has it's own advantages in certain circumstances. Linemode is the most efficient to use (because you bother telnet the least, and transmit to the network the least - which produces a savings in the overheads required), but you can't define your own end-line character and can't effectively use programs that act on single keypresses. Msgmode bothers telnet for every character, but won't transmit to the serving host until you type the end-line character. This reduces the overhead in writing to the network, but causes lots of overhead in bothering telnet with input characters. Charmode is, then, the most gluttonous of all three as it bothers telnet for each character and then writes single characters to the network.

However, one can take a slightly different view of the whole matter. Many computer systems do character translations that user telnet does not do (for example, making control-c come out looking like ^C) which make charmode the least confusing mode to use. Since linemode and msgmode are saving up your characters until you type an end-line, the local telnet must do the echoing and you lose any such character translations. The loss of this type of facility may make one feel isolated from the foreign computer. This isolation from the host computer makes the whole connection feel somewhat 'rubbery' and uncontrollable - which lots of people find disconcerting, if not downright annoying.

In summary, the mode you choose is purely a matter of personal preference. If the machine you connect to does not require charmode, linemode is your best bet because it will cause less load on your local system. However, if you use a system like a TENEX, you will probably be miserable if you're not in charmode. As a matter of fact, there is a convience mode created especially for TENEX systems called (what else) 'tenex'. This mode is a faster way to say 'charmode' with 'echo off' (so the TENEX does the echoing, funny translations and all).

2.3.3 Telnet Protocols and Options

The user telnet program communicates with the server over a logical network connection. User telnet, located in the computer you are connected directly to, provides you the interface to the network. Commands to user telnet, as explained earlier, control either your interface to the network or the interface from the network to your host computer (thru the telnet server located

there).

Of course, the server and user telnet programs may exchange status information, alert each other to special conditions, or make special requests about the connection to each other. In order to accomplish these tasks both programs must have an agreed upon set of rules - a protocol - that each follows to understand the other's special messages. The full telnet protocol is quite versatile and easy to use. Unfortunately, most telnet programs (user and server alike) fail to implement the entire set of protocol requests due to many and varied reasons.

Within the protocol, there are several sub-divisions. If we were to draw a line, it would distinguish between 'command' protocol requests and 'negotiation' protocol requests. The former is a unilateral case - one program informs the other that it is to take a specific action; the latter, on the other hand, requires that both programs 'agree' on some action. The debate between telnets which follows when one asks for something and the other must agree is termed 'option negotiation'. Often one side is prepared to perform some option and requests (usually at the insistence of the user) that the other side conform to it's desires. The other side then has the ability to accept or decline the new request. If it's declined, the requestor may not go ahead and do it anyway, as confusion is sure to ensue. This version of telnet will send all of the command requests, but fails to negotiate the options well[6].

What are the command requests for? Basically, they allow you to control the program that you're running at the host computer with greater accuracy that would otherwise be possible. When you send a protocol message, you are telling the telnet at the remote host (the server) that it must do something. Commonly you would tell the server to interrupt the program you have running or throw away the output from it. These functions are accomplished with the telnet ip (interrupt process) and ao (abort output) respectively. Other functions of this type include the synch sequence (used to synchronize the server and user telnets when some trouble is experienced with the connection), and the are-you-there function (which asks the foreign host if it is still up and running).

---

[6] Unfortunately, such requests must be sent literally via the send command, which requires knowing the numeric values of the telnet protocol commands. All would be fine if only Unix had a viable IPC facility, as each process cannot know what the other is doing.

Option negotiations are considerably more exotic. Currently there are about 21 telnet options defined in the network community, and the list grows continuously. Among the options are: transmission of binary information, setting of horizontal and/or vertical tab settings (for those terminals that have hardware to support such), and providing for the serving telnet to control the manner that characters you type are echoed to you by your user telnet. The most common, of course, is plain and simple controllin of echo. Either telnet can request that the other (or itself) tell you what you typed. If you're on a half-duplex terminal, neither of the telnets are echoing what you type (the terminal is doing it for you). Unfortunately, user telent does not implement most options, due to lack of IPC.

2.4 Illustrative examples

This section will present some examples of using telnet. The examples are taken from typescripts of actual sessions at the terminal. However, they have been edited in some cases to truncate outputs which would not fit in the document and which do not add materially to the usefulness of the example. When a line starts with a "%", the rest of the line was typed to a Unix shell. Comments which have been added later are bracketed with "[[" and "]]".

2.4.1 Simple Connection

The following illustrates the minimal commands necessary to make a network connection.

```
% telnet
 Unix user telnet -- Version 4.0 (11.37)
 ^ esc $
 Escape character is $
 $ : will turn off messages to prevent confusion
 $ : unix treats lines starting with colon blank as comment
 $ : while not connected, telnet puts esc char as prompt
 $ : when connected to other host, i have to type it
 $ mesg
was y
 $ : will now try connect to same system
 $ connect ill-unix
 Attempting connection to ill-unix
 Interrupted - abort attempt
 $ : got tired of awaiting timeout, hit rubout
 $ : try a different host, this time a tenex
 $ tenex
 $ connect bbn
 Attempting connection to bbn
 Connection open (charmode)
```

```
 BBN-TENEX 1.34.13, BBN-SYSTEM-C EXEC 1.54.55
@LOGIN KELLEY
 ?
@LOGIN
(USER) KELLEY
(PASSWORD)
(ACCOUNT #) 111641
 JOB 30 ON TTY37 9-May-78 15:16
 PREVIOUS LOGIN: 9-May-78 01:16
 TENEX will go down Wed 5-10-78 0400 til Wed 5-10-78 0800
 due to scheduled hardware work by SPEAR
@$: now whatever i type goes to bbn, unless preceded by $
 $
DIR

   <KELLEY>
 /UNDELIVERABLE-MAIL/.wyliewUSC-ISIC;1
 MESSAGE.TXT;1
 ]ARCHIVE-DIRECTORYE.;1

@$: the content of those files is of little interest
 $
LOGOUT
 TENEX will go down Wed 5-10-78 0400 til Wed 5-10-78 0800
 due to scheduled hardware work by SPEAR

KILLED JOB 30, USER KELLEY, ACCT 111641, TTY 37, AT  5/09/78 1545
  USED 0:0:19 IN 0:28:28
$close

Connection Closed
 Linemode
 $ : when the connection is closed you go back to linemode
[[There is no logical reason for changing escape character
  in this circumstance, just did it for example. Next part
  is simple connection to another site, but using timeout
  parameter in case they are not responding well.        ]]
 $ connect ccn -t 10
 Attempting connection to ccn
 Connection open (linemode)
UCLA CCN 360/91 SERVER TELNET
VERSION 4.0 12/76 (NEW TELNET PROTOCOL)
ENTER COMMAND OR 'HELP':
help

Commands available are:

Service--Description

RJS.....EbCDIC remote job submittal service.
ARJS.....ASCII remote job submittal service.
```

```
TTYRJS...Alternate ASCII RJS for a model 33 TTY.
BBOARD...Bulletin board notices of general
         interest to CCN users.
TSO.....Access to IBM TSO time sharing system.
NETSTAT..Give status of users connected to CCN's
         NCP. Lists status of the protocols.
HELP s...Produces information about a service
         where 's' is a service name.
END.....Disconnect from CCN's server telnet.
QUIT.....Same as END.
CLOSE....Same as END.
OPR.....Send messages to the operator. Note
         for reply use TSO MSG or NET MAIL.

Commands except for HELP have no operands but
must be followed by a CR/LF. Any nonambiguous
abbreviation for a command is acceptable. For
further information about CCN services,
call (213) 825-7548.

ENTER COMMAND OR 'HELP':
$: it is rather nice of them to keep
 $
$: help stuff on left side of screen
 $
$close

Connection Closed
 $ $
 $ : now that i am not connected i do not need the escape char
 $ : my close did the same as their end would have
 $ status

type  hostname  user-id p-id fileid usr state  status
---- ---------- ------- ---- ------ --- ------ ----------------------
RECV    anyhost     NCP    0  40440 NCP  LIMBO Unknown
RECV bbn-tenexb   marty 28736  37440 USR   OPEN Data Wait
SEND bbn-tenexb   marty 22831  37440 USR   OPEN Just got ALL
RECV bbn-tenexb    walt 24071  40120 USR   OPEN Data Wait
SEND bbn-tenexb    walt 18450  40120 USR   OPEN Just got ALL
SICP    anyhost     NCP    0  37730 NCP LISTEN Unknown
RECV bbn-tenexb   marge 16163  40000 USR   OPEN Data Wait
SEND bbn-tenexb   marge 22280  40000 USR   OPEN Just got ALL
SICP    anyhost     NCP    0  40210 NCP LISTEN Unknown
SICP    anyhost     NCP    0  40370 NCP LISTEN Unknown
 $ conn dti -t 10
 Attempting connection to dti
Open fails: I/O error
 $ : that is the condition you get on timeout of attempt
 $ tenex
 $ conn bbn
```

```
 Attempting connection to bbn
 Connection open (charmode)

 BBN-TENEX 1.34.13, BBN-SYSTEM-C EXEC 1.54.55
@$status

type  hostname  user-id p-id fileid usr state  status
---- ---------- ------- ---- ------ --- ------ ----------------------
RECV    anyhost     NCP    0  40440 NCP  LIMBO Unknown
RECV bbn-tenexb   marty 28736  37440 USR   OPEN Data Wait
SEND bbn-tenexb   marty 22831  37440 USR   OPEN Just got ALL
RECV bbn-tenexb    walt 24071  40120 USR   OPEN Data Wait
SEND bbn-tenexb    walt 18450  40120 USR   OPEN Just got ALL
SICP    anyhost     NCP    0  37730 NCP LISTEN Unknown
RECV  bbn-tenex    karl 15917  40100 USR   OPEN Data Wait
SEND  bbn-tenex    karl  225  40100 USR   OPEN Just got ALL
SICP    anyhost     NCP    0  40210 NCP LISTEN Unknown
SICP    anyhost     NCP    0  40150 NCP LISTEN Unknown
 $
$: note that i am now owner of two simplex connections
 $
```

2.4.2 Specific Sockets

The following illustrates a connection to a specific socket.

```
[[The capability of connecting to specific sockets allows one
  to access special services on other sites. The following is
  not recommended as a way to send mail, because a program
  called sndmsg usually does it for you and does a better job.
  The purpose is to illustrate specific socket connection. Do
  not connect to a socket without knowing precisely what that
  side is expecting to appear on the line (else bad results).]]
Last login Tue May  9 16:04:50 1978
You have old mail
13 users @ Tue May  9 16:15:46 1978
% telnet bbn -s 3
 Attempting connection to bbn
 Connection open (linemode)
300 BBN-TENEX FTP Service 2.9.0 %57 at Tue 9-May-78 17:20-EDT
mail kelley
350 Type mail, ended by a line with only a "."
this is a test message, snt as part of example session
256 Mail completed successfully.
^close

Connection Closed
[[Now that the connection is closed, we can connect in the normal
  way to read the mail we just sent.                          ]]
 ^ con bbn
```

```
 Attempting connection to bbn
 Connection open (linemode)

 BBN-TENEX 1.34.13, BBN-SYSTEM-C EXEC 1.54.55
@^tenex
 Charmode
LOGIN KELLEY  111641
 JOB 38 ON TTY37 9-May-78 17:22
 PREVIOUS LOGIN: 9-May-78 15:58
 TENEX will go down Wed 5-10-78 0400 til Wed 5-10-78 0800
 due to scheduled hardware work by SPEAR
 [YOU HAVE NEW MAIL]
@READMAIL
READMAIL 2E(15)
TYPE ? FOR HELP
*
Mail from ILL-UNIX rcvd at 9-May-78 1720-EDT
this is a test message, snt as part of example session
@LOGOUT
 TENEX will go down Wed 5-10-78 0400 til Wed 5-10-78 0800
 due to scheduled hardware work by SPEAR

KILLED JOB 38, USER KELLEY, ACCT 111641, TTY 37, AT  5/09/78 1725
  USED 0:0:3 IN 0:3:16
^close

Connection Closed
 Linemode
 ^ bye
%
```

2.4.3 Connecting to Another Unix

The echo off command can be used to hide a password; the stty_echo command at a remote Unix system can be used to turn off remote echo.

```
% con dti
 Attempting connection to dti
 Connection open (linemode)

>>> ENFE Server Telnet <<<
      New Protocol

Digital Technology Unix
Login: kck
kck
Password: ^echo off

% echo on
```

```
stty -echo
stty -echo
% who am i
kck      ttyX May 10 20:45
% cat /etc/motd
26Apr78        Ok, guys.  Time to clean house.  Disk space is really low
        on rp5.

30Apr78 HEY, FOLKS, We're down to 1500 blocks on rp5....throw things out
        or they will disappear mysteriously.
% ^cat /etc/motd

10May78 HASP Off while debugging subsystem problems.

        DOWNTIME: possibly thurs 5/11 0930-???? for new AED disk install

        Please: NO net-mail to CMUA until further notice (Balocca)

        Congratz to Diane, Tom, & Michael Milke (7#6oz: 1030 Thurs.)

%
```

2.4.4 Multiple Connections

The capability to work with two open network connections does not really exist on this system. However, because Unix will allow the user to run a program as a subshell, it is possible to put one connection on "hold" while running a second embodiment of telnet. This is not advised for the Unix novice.

```
[[The next example is only for people familiar with unix. Since
  the unix telnet program allows the user to run any program as
  a forked off shell, and since telnet is itself a program, then
  if I know the full pathname of the unix telnet program...     ]]
$: i can get a second connection going by the following trick
 $
$/usr/bin/telnet bbn
 Attempting connection to bbn
 Connection open (linemode)

 BBN-TENEX 1.34.13, BBN-SYSTEM-C EXEC 1.54.55
[[Now of course I cannot in any way get the attention of the
  first telnet program I was running until I completely exit
  this one.                                                     ]]
@login kelley
LOGIN KELLEY
(PASSWORD) applesauce
[[I forgot to turn echo off so my password showed up. The
  command ^tenex should have preceeded the connect command.
  However, since I am editting in these comments, what
```

really appeared there has been changed to applesauce.]]

```
(ACCOUNT #)
111641
 JOB 2 ON TTY50 9-May-78 15:53
 PREVIOUS LOGIN: 9-May-78 15:16
 TENEX will go down Wed 5-10-78 0400 til Wed 5-10-78 0800
 due to scheduled hardware work by SPEAR
@^: now i am in a subordinate shell running another
 ^
^: copy of telnet, no relation to the first, soon

 AUTOLOGOUT
KILLED JOB 26, TTY 45, AT  5/09/78 1559
  USED 0:0:1 IN 0:3:3
 ^
 Connection aborted
^: i should see (I just did see) evidence that first
 ^
^: connection attempt failed cause i did not login
 ^
^: might
 ^
^: now when i close this connection and say bye i will be
 ^
^: back to the initial embodiment of telnet
 ^
logout
LOGOUT
 TENEX will go down Wed 5-10-78 0400 til Wed 5-10-78 0800
 due to scheduled hardware work by SPEAR

KILLED JOB 2, USER KELLEY, ACCT 111641, TTY 50, AT  5/09/78 1602
  USED 0:0:2 IN 0:3:14
^bye

Connection Closed
 $
[[Now I am back to the original telnet session, but the connection I
  had open to bbn is closed because of the autologout.          ]]
```

2.4.5 Using Send Command

The following session is an example of the necessity and use of the send command.

```
% : session to illustrate usefulness of send command
% con bbn
 Attempting connection to bbn
 Connection open (linemode)
```

```
 BBN-TENEX 1.34.13, BBN-SYSTEM-C EXEC 1.54.55
@^tenex
 Charmode
LOGIN KELLEY  111641
 JOB 27 ON TTY37 12-May-78 11:59
 PREVIOUS LOGIN: 9-May-78 17:22
@SEN^H ?
@SNDMSG

To (? for help): KARL@ILL-UNIX

cc (? for help):

Subject: EXAMPLE MESSAGE

Message (? for help):
WHILE TYPING THIS EXAMPLE IN UNIX, THE AT SIGN IN THE  TO
LINE HAD TO BE
**PRECEEDED BY A BACKSLASH. EVEN IN CHARMODE THE UNIX HANDLING
OF SPECIAL
** CHARATERS  SRETAACTERS IS IN EFFECT.

HOWEVER, IT WOULD SEEM THAT
** ON THE BBN SYSTEM THE LINES ARE ALL BEING TREADED  DETESSD
AS THOU
**GH I WAS NOT PUTTING CARRIAGE RETURNS AT THE END.
TO PRLVENT INTERFEREN
**CE WITH ITS SPECIAL FUNCTION IN THIS SCRIPT-PRODUCING
MODE, I WILL SEN
**D THE REQUIRED CONTROL-Z USING THE SNED COMMANDDNAMMOC DENE
**ND COMMAND.
^send /z
^Z

Q,S,?,carriage-return: Send

KARL at ILL-UNIX -- ok
@LOGOUT

KILLED JOB 27, USER KELLEY, ACCT 111641, TTY 37, AT  5/12/78 1216
  USED 0:0:6 IN 0:17:33
^close

Connection Closed
 Linemode
 ^ bye
% msg

10 messages in ./mailbox

For help type ?
```

```
<- type 10

                Message 10 has 691 bytes, 17 lines
Timestamp: Network mail from host bbn-tenex on Fri May 12 11:14:39
Date: 12 May 1978 1215-EDT
From: KELLEY at BBN-TENEX
Subject: EXAMPLE MESSAGE
To:   KARL at ILL-UNIX

WHILE TYPING THIS EXAMPLE IN UNIX, THE AT SIGN IN THE  TO
LINE HAD TO BE PRECEEDED BY A BACKSLASH. EVEN IN CHARMODE THE UNIX HAND
OF SPECIAL CHARACTERS IS IN EFFECT.

HOWEVER, IT WOULD SEEM THAT ON THE BBN SYSTEM THE LINES ARE ALL BEING T
AS THOUGH I WAS NOT PUTTING CARRIAGE RETURNS AT THE END.
TO PREVENT INTERFERENCE WITH ITS SPECIAL FUNCTION IN THIS SCRIPT-PRODUC
MODE, I WILL SEND THE REQUIRED CONTROL-Z USING THE SEND COMMAND.
-------

--- Network mail transmission ended on Fri May 12 11:14:43  ---
<- quit
% : my typing skill was not too impressive in this message.
% : the **in front of lines produced in sndmsg indicate where they
% : forced the line to end.
% : However, note that in the resultant mail, lines end correctly.
% : end of example
%
% telnet
 Unix   user   telnet   --   Version   4.0   (11.37)    ^    he
he              [[abreviation of help help]]
          help      [<keyword>]
                Invokes the help function to tell you some  useful
                things   about  the  keyword  specified.   If  the
                keyword  is  unrecognized,  or   no   keyword   is
                specified, instructions for using the help command
                are       given.          ^          help       garbage
intro                                                 [[Neither key is
a command, so                        you get information about  com-
mands.]] Introduction to user telnet help functions.    Type:
    help            to get this
    help  <cmd>     for a description of specified command
    commands        for a list of the commands available
    hosts           for a list of the possible  host-names  avail-
able Several shorthand notations are used in the command descrip-
tions, especially  in the line giving the valid syntax  for  each
command.  These are:
  a) Optional items are enclosed in square brackets ("[", "]")
  b) Command syntax lines list the  entire  command  -  prefixes,
     parameters, and all.
  c) Parameters are usually specified by  mnemonics  enclosed  in
     angle brackets ("<", ">")
```

d) Unimplemented commands are enclosed in double curly-braces ("{{", "}}") status bbn

bbn will be dead for an unknown period due to...[[ed.]] ^ hos [[output of this command has been truncated on right and shortened.]]

accat-tip^	acl^	afwl^	ai^
aim^	ames-67^	ames-tip^	anl^
argonne^	arpa-dms^	arpa-tip^	asl^
bbn-tenexb^	bbn-tenexd^	bbn-tenexe^	bbn-testip^
bbnd^	bbne^	bnl^	cca^
ccn^	cctc^	cincpac-tip^	cmu-10a^
cmu-10d^	cmua^	cmub^	cmud^
dms^	docb-tip^	eglin^	etac^
gunter-elf^	gunter-tip^	gwc-tip^	harv-10^
i4-tenex^	ill-nts^	insecurity^	isi^
isic^	isie^	ka^	kl^
lbl^			
ll-asg^	lll-unix^	lon-eps-gate^	london^
london-tip^			
mit-ai^	mit-dms^	mit-multics^	mit-tip^
mitre-tip^			
multics^	nbs^	nbs-10^	nbs-tip^
ncc-tip^			
norsar-tip^	nosc-cc^	nosc-sdl^	nrl^
nsrdc^			
nswc-dl^	nts^	nyu^	office-1^
ot-its^			
parc^	parc-maxc^	pentagon-tip^	radc-tip^
rand-tip^			
rutgers-10^	sail^	satnet^	scrl-rsx^
sdl^			
sri^	sri-ai^	sri-ka^	sri-kl^
su-ai^			
su-tip^	sumex-aim^	ucla-ats^	ucla-ccn^
ucla-security^			
uk-ics^	usc-isi^	usc-isic^	usc-isie^
usc-tip^			
utah-tip^	utexas^	wpafb^	wpafb-afal^
xerox-parc^			

110 alive (75 distinct) / 96 dead (74 distinct)

^ : now to try a few things that are prohibited, and watch the results.

```
^ set
Unknown set command parameter
^ send
No open connection for sending
^ ec
Command requires a prefix
^ disp
Character mapping not implemented
```

```
^ qu % con
Unix user telnet -- Version 4.0 (11.37)
^ tenex
^ con sri-kl
Attempting connection to sri-kl
Connection open (charmode)

SRI-KL, TOPS-20 Monitor 1018(114)
System shutdown scheduled for Thu  9-Mar-78 23:00:00,
Up again at Fri 10-Mar-78 04:00:00
There are 81+7 jobs and the load av. is  8.03 @attach (user)
?  @attach nobody  ?  @passwd
?  @^escape :
Escape character is : sys
Thu 9-Mar-78 15:50:45  Up 14:14:45
82+7 Jobs   Load av   7.49  13.96  16.04

System shutdown scheduled for Thu  9-Mar-78 23:00:00,
Up again at Fri 10-Mar-78 04:00:00

Job  Line Program  User       Foreign host
  3   Det  RSSER   Dev-rsexec
  4   Det  DC-4/1  Dc-203
  5   123  SOS     Berson
  6    64  EXEC    Guthary
  7   Det  OJMLAV  Jdemon
 13   114  LISP    Slocum
 14   142  EXEC    Garvey
 15   232  LISP    Appelt     (SU-AI)
 16    57  MACLSP  Elspas
 17    62  EXEC    Moriconi
 18    35  TVEDIT  Hobbs
 19    10  EXEC    Hendrix
 20     4  EXEC    Agin
 22   122  EXEC    No login
 23   174  MSG     Bair
 24     5  TVEDIT  Hodel
 25   173  TVEDIT  Nilsson
 26   Det  NLS     Lieberman
 27   107  EXEC    Seitz
 28   162  LISP    Weinstock
 29   230  NLS     Feedback   (ARC-RD)
 30   145  MSG     Jones
 31   121  COST    Stroke
 32   Det  FTP     Hysmith
 33    37  EXEC    Grosz
 34   Det  EXEC    Thayer
 35    12  EXEC    Klh
 36    66  EXEC    Agin
 37   125  EXEC    Robinson
 38    24  EXEC    Hillhouse
```

39	43	EXEC	Melling
40	115	LISP	Wolf
41	105	EXEC	Sagalowicz
42	3	TVEDIT	Untulis
43	76	BSYS	Seitz
44	151	EXEC	Green
45	71	NLS	Heckel
46	165	TV	Lamport
47	Det	DIABLO	Guthary
48	116	EXEC	Katemopoulos
49	160	EXEC	Williams
50	53	LISP	Konolige
51	240	EXEC	No login (RUTGERS-10)
52	146	DIABLO	Lamport
53	7	MM	Mmcm
54	72	EXEC	Hopper
55	51	TVEDIT	Mellar-smith
56	167	EXEC	Sacerdoti
57	144	SAIL	Quam
58	70	EXEC	Lrobinson
59	100	EXEC	Barrett
60	Det	EXEC	Blean
61	234	EXEC	Vannouhuys (ARC-RD)
62	156	LISP	Wilber
63	Det	EXEC	Blean
64	231	NLS	Poggio (ARC-RD)
65	22	EXEC	Adrian
66	Det	FTPSER	No login
67	Det	EXEC	Jrobinson
68	111	TECO	Novak
69	Det	EXEC	Thayer
70	164	TVEDIT	Neumann
71	Det	TEST	Hedrick
72	21	SNDMSG	Torres
73	45	EXEC	Walker
74	Det	PRVPRC	Landes
75	242	FTPSER	Hedrick (RUTGERS-10)
76	61	EXEC	Bolles
77	15	EXEC	Gene
78	120	EXEC	Mathis
79	170	EXEC	Diane
80	235	FTPSER	Outjournal (OFFICE-1)
81*	237	EXEC	No login (ILL-UNIX)
82	236	EXEC	No login (USC-TIP)
83	6	LISP	Gaschnig
84	Det	COST	Stroke
85	40	EXEC	Mcghie
86	117	EXEC	Roy
88	241	DUNGEO	Meradcom (MELVOIR)
89	2	EXEC	Heathman
91	102	LISP	Pease

```
  92   233  EXEC     Nelc       (NOSC-SDL)
  93   Det  EXEC     Norton
  94   245  NLS      Netinfo    (USC-ISIB)
  95    77  EXEC     Gleason
  96   Det  EXEC     Agin
  97    52  EXEC     Bolles
  98   243  EXEC     No login    (SU-AI)

   1   202  PTYCON   Operator
   2   203  DAEMON   Operator
   8   205  BATCON   Operator
   9   206  LPTSPL   Operator
  10   Det  BSYSPL   Operator
  11   210  CERBER   Operator
  12   211  FUSS     Operator @:close

Connection Closed
 Linemode
 : esc +
 Escape character is +
 + elin
 Endline character is '
 + bye % con
 Unix user telnet -- Version 4.0 (I1.37)
 ^ con ill-nts -s 23
 Attempting connection to ill-nts
 Connection open (linemode)

No resources.
 Connection aborted

 Unable to transmit last 2 characters
 ^ con ill-nts
 Attempting connection to ill-nts
 Connection open (linemode)

>>> ENFE Server Telnet <<<
      New Protocol

Digital Technology Unix Login: jsg Password:

/mnt/jsg %

% ^esc
 Escape character is ^ ^con
 Connection still open


%: commands are repeated twice because both sites are echoing.  %
esc + esc + esc: not found % ^esc +
 Escape character is + +con ill-nts -s 3
```

```
 Connection still open


% cat mailbox % %: no mail, let's make some with telnet and ftp.
%: note the one conn max, so have to reinvoke telnet by full
pathname, %: because the telnet and shell con commands have the
same name. % +/usr/bin/con ill-nts -s 3
 Attempting connection to ill-nts
 Connection open (linemode) 300 NFE Unix Server FTP mail jsg 350
Enter Mail, end by a line with only '.' this is fake netmail that
we'll look at in a minute. note that there are now two connec-
tions open to ill-nts, the original telnet connection, and the
child that telnet forked off (/usr/bin/con) which is another tel-
net to ill-nts to the normal ftp server socket. this exercise
can be confusing. changing the escape character is recommended
in the case of more than one connection open. 256 Mail Delivered
^cl

Connection Closed
 ^ : still in the sub-telnet. this will get passed to the
 ^ : shell as it is not a telnet command. the shell will
 ^ : then ignore it, as the colon makes it a comment.
 ^ by
 +


% who am i who am i jsg     ttyX Mar  9 01:17 % cat mailbox cat
mailbox --- Network Mail from host host-239 on Thu Mar 9
01:18:49 --- this is fake netmail that we'll look at in a
minute. note that there are now two connections open to ill-nts,
the original telnet connection, and the child that telnet forked
off (/usr/bin/con) which is another telnet to ill-nts to the nor-
mal ftp server socket. this exercise can be confusing. changing
the escape character is recommended in the case of more than one
connection open. % cat /dev/null>mailbox cat /dev/null>mailbox %
: why leave it laying around?? : why leave it laying around?? %
: double echo occurs because the server at ill-nts is half du-
plex. : double echo occurs because the server at ill-nts is half
duplex. % +clos

Connection Closed
 + con -h 2114
 Attemptin; connection
 Connection open (linemode)

>>> ENFE Server Telnet <<<
      New Protocol

Digital Technology Unix Login: +disc

Connection Closed
```

```
 + con -h 0114 -t 1
 Attempting connection
 Connection open (linemode)

>>> ENFE Server Telnet <<<
      New Protocol

Digital Technology Unix Login: +: too bad, thought the small con-
nection timeout would blow
 + +: the connection before it got going.  the net must be busier
to
 + +: prevent that from happening.
 + +: thats all for now
 + +cl

Connection Closed
 + bye
```

3 User Telnet Maintenence Guide

Up to this point the description of the telnet program(s) has been fairly general and not really told of the implementation details. Now we address those details.

3.1 Principles of Operation

As stated earlier, user telnet is primarily a character shuffler. The available terminal modes for user telnet (linemode, charmode, msgmode) dictate how this shuffling is done. Linemode, as the name implies, is a line-at-a-time mode; characters are buffered by the system and given to user telnet when the user types a line terminator. Charmode does character-wise output to the net, collecting and buffering characters from the user terminal only for as long as it takes for the last network write to complete. Msgmode (message mode) is a cross between the two: buffering characters one at a time, but sending them only when the (user-settable) telnet endline character is typed.

Due to the fact that Unix read and write system calls are blocking (that is, they will not return until the read or write is completed), the implementation of user and server telnets requires two processes for each connection. One process is used to read from the net and write to the terminal (or pseudo-terminal for server), while the other reads the terminal (pty for server) and writes to the net[7]. The process of user telnet that reads the terminal is also responsible for interpreting any commands the user may type to telnet. The process (for both user and server) reading from the net is responsible for interpreting the telnet protocols received from the foreign telnet, and communicating state changes induced by negotiations to it's companion. Accomplishing this Inter-Process Communication (IPC) function is difficult and cumbersome[8] within the frame of standard Unix.

---

[7] There is no reasonable way, within the standard Unix system or the current Illinois Unix, to get around this problem. However, the addition of an enhanced Inter-Process Communication system (specifically events) would allow one process to wait for whichever system I/O operation completed first. Further discussion of events may be found in the later appendix on IPC.

[8] Standard Unix provides only the signal IPC mechanism. This mechanism allows no data to be sent between processes, only an indication that something has occurred. Further, there are a reasonably small number of signals, which means that you can only transmit a very small set of messages that something has occurred. But the biggest

3.1.1 Process Structure

In order to allow complete full-duplex interaction, there must be two processes for each telnet connection. One to read from the net and write to the terminal; the other to read the terminal and write the net. This state of affairs is caused by Unix system calls blocking the calling process until they complete - it would be unacceptable to have the terminal unresponsive when waiting for the net I/O to complete, and vice-versa.

Towards this end, when a user runs telnet, he is interacting with the process that reads from the terminal. Upon successful completion of an open on the network, this process does a fork and the child exec's the companion process which will read from the net and write to the terminal.

From this point on, all of the user's terminal input is passed to the net unless the line begins with the escape character. In this case, telnet attempts to interpret the command and complete the requested action. If the command is ambiguous, no action (save an error message) occurs - on the other hand, if the command is simply unknown, telnet will do the fork/exec sequence to hand the command to a shell (using the shell '-c' switch) for execution. As an added convience, a line beginning with an escape character, but having no command on it, will cause telnet to fork/exec a shell for the user to control from the terminal.

This fork/exec sequence is normally done by a routine which takes pains to leave the connection features unaltered. Before the fork/exec is done, the terminal mode bits are reset to what they were when the user entered telnet, and telnet is suspended[9] until the child process has died. Then this routine resets the terminal modes to their state just prior to executing the command, and proceeds (see the description of the 'execute' routine).

---

trouble with signals is the unfortunate fact that a system i/o call will be aborted, with it's fate unknown, when a process is sent a signal.

[9] This suspension is done via a wait system call, the terminal resetting is accomplished via a stty system call (having done a gtty system call and saved it's results when telnet is first entered).

This procedure for executing auxiliary programs (next section) and Unix commands can have interesting and confusing (for the beginner) effects in some cases. For example, no matter how hard you try to reset the terminal modes via the stty command it just won't appear to be working. In actuality, the requested change is indeed being performed, and then promptly stepped on when the child dies. It is also worth mentioning that the companion process to handle the other side of the full duplex connection is not executed by this routine. If it were, the parent would wait for it to die before accepting input from the terminal to pass to the net.

3.1.2 Auxiliary Programs

Some of the telnet commands will, in turn, cause other subsidiary programs to be executed via this fork/exec sequence. Among such functions are the 'hosts' command (executing hosts), the 'help' command (help), and the 'status' command (either hosts or netstat)[10]. Of course, the companion process (/etc/usrtelnetin) is executed whenever a connection is opened, but being the result of a rather special command, it is treated in a different manner upon execution (see the description of routine 'netopen'). In the horrible event that any of these programs have vanished or the system is out of processes, an error message is produced.

3.1.3 Terminal Modes and Character Buffering

As stated several times before, user telnet will allow use of one of several terminal modes. Basically, these modes control either echoing, or how the characters the user types on the terminal are handled. Echo control is quite simple; the user may tell telnet to turn the local echo on or off, regardless of the state of the server on the other end. Character handling modes are more complex because there are more to chose from, and their effects are more widespread.

Telnet allows three character handling modes: linemode, msgmode, and charmode. Linemode is implemented as Unix cooked mode - utilizing the system-level editing provisions; charmode and msgmode are implemented via Unix raw mode and differ only in when data is sent to the net. For charmode, each character is sent as soon as it is received by user telnet; msgmode waits for a user-settable endline character (see the 'eline' command description) to be typed before transmitting characters to the

---

[10] The IMP/interface status program specified (netstat) may vary from installation to installation.

net.

When in linemode, the system buffers up input characters until a (system-defined) line terminator is typed. User telnet is then awakened and given the buffer full of characters. For the modes that use Unix raw mode, telnet reads characters as they are typed, and then buffers them until they are transmitted to the net. When a user-settable escape character (see the 'escape' command description) is the first character on a line, the entire line is buffered by telnet (regardless of the mode) and then interpreted as a command.

3.1.4 Character Mapping Facility

Ultimately, telnet may provide full duplex character mappings thus: establishing correspondences between characters received from the net and arbitrary (user-defined) character strings to be typed on the terminal, as well as mapping keys typed on the terminal to arbitrary (user-defined) character strings to be sent to the net. However (again due to poor Unix IPC facilities), the mapping of chars from the net is not implemented in any form at the time of this writing. Mappings from keys to character strings are currently implemented via a table-lookup scheme. The table is not prepared by telnet itself, but by another auxiliary program: mkcharmap[11]. As one of it's initial actions, telnet searches in the user's home directory for a file by the name of 'character_set', which is loaded immediately if found.

The user may also request that a character map set be loaded via the load command, which takes, as an optional argument, a pathname of the file to load. There is no checking of any form on the filename, so it is possible for one to load any file. If one does not load a file produced by mkcharmap, however, very unusual things may transpire.

3.1.5 Telnet Option Negotiation

Currently, one may send negotiated telnet options to an open connection only via the 'send' command. This requires that one know the numeric value (octal or decimal) of the correct protocol string for the desired action. Again, this facility has not been expanded due to unsatisfactory IPC facilities on Unix.

---

[11] The mkcharmap program leads the user thru the steps for creating a character map set. It lacks several features that may be deemed desirable, and causes some trouble for the buffering scheme (with regards to echoing).

As the code now stands, it is possible to set up options before a connection is opened that telnet will negotiate when a connection is first opened. In general, these requests may only come from the program logic, as the command table lacks the keywords specifying the various options. The hooks are in the code to completely do negotiation, but (yet again) the Unix IPC problems prevent reasonable implementation. Refer to the 'negotiate' routine, and the sections on IPC and possible enhancements for more discussion of option negotiations.

3.2 Major Data Structures

Most of the data structures of user telnet are C structs or constants. For the constants, please refer to the parameter files cited above (usrtelnet.h, telnet.h, netopen.h) and assorted others that are definitively listed in the source of user telnet; structures will be sketched here.

3.2.1 Command Table Structure

Seemingly, the most important function of user telnet is the interpretation of the commands the user types on his/her terminal. These commands are looked up in something called the command table (com_table) to determine what effects they will produce. Each entry in the command table array is a structure named com_entry which contains : char *com_name (text - name of keyword), int com_param (arbitrary data item used in invoking the associated C function), int com_useflag, and int (*com_proc)() (ptr to the activated proc). The sequence of events is roughly this: a routine invokes getcomm to determine if the suspected command is indeed a command; getcomm returns a pointer to the entry if it is, and the routine is then free to take whatever action it desires - the com_entry provides it with all information needed. The useflag field in the entry contains a bit pattern which describes what other commands a command may be prefixed by, whether it requires a prefix, etc. The param is a command dependent parameter to the routine for servicing the command. For example, the routine helper gets invoked for 'help', 'commands', and others, and the parameter tells them apart. The last entry in the command table is a dummy element composed of all zeros. Its used to terminate searches, and as a dummy pointer when getcomm detects an ambiguity.

3.2.2 The Openparams Structure

To make opening the net easier, a standard structure for specifying network open parameters is available (see netopen.h). The various exotic things that this struct can be used for are not included in the scope of this document, but a few will be mentioned. The o_fskt field specifies the socket to which the connection will be made. This is 1 (one) by default, and should

be changed only in cases like: the well known server telnet socket changes across the net. The default number of seconds that user telnet will wait before deciding that a connection has not been accepted is set by the o_timeo field. If the connection has not been completed by that amount of time, the NCP returns an error indication.

Four of the fields in the openparam struct may be set as parameters to the 'connect' command. These are: the number of seconds before a connection attempt times out, the foriegn socket that a connection will be attempted to (included for experimenting with new telnets, establishing a connection to an ftp, etc.), the nominal message allocation, which is not particularly useful for telnet, and the host number which may be specified when the name of a host is unknown but the number is handy.

3.2.3 Character Buffering and Mapping

Included in this category are: the character buffer (a getc-like structure called instruct), and various character mapping tables. Instruct contains a reasonably small (currently 80 byte) buffer for characters, a count of the number of characters remaining in the buffer, a pointer to the next one, and the crossover to the character maping facility - a pointer into the string table (stringtab) if a mapped char is being expanded. The string table contains character strings which will be sent when the user types a character which has been defined to be mapped. To determine if a character is mapped, we consult yet another table - the map table (maptab). This one has 128 (octal 0200) entries (one for each ascii character) which are set to indicate the presence of mapping on that character, if it is mapped. For clarification, the reader is advised to read the section on the modules map_echo and getinpc, then reread this section.

3.3 Major Modules

This section and the next contains all the modules that exist in usrtelnet, and brief descriptions of how they work to aid in understanding the program. This list is brief in order to give an impression of the general module interactions without obfuscation by messy details; see the code for really up to date information. The modules are not arranged as they appear in the code (for the most part); this is an attempt to put them in a hierarchical order.

3.3.1 usrtelnet (main)

The main program of telnet provides necessary initializations, and then loops around deciding what to do with characters the user types. If (s)he types a telnet command, the command processor is invoked as the dispatcher. Of course, when no net connection is open every line is interpreted as a command; with a connection open only those line beginning with the escape character are treated as commands. Note that initialization could include initiating a connection, with the arguments on a command line interpreted as if a telnet connect command had been typed.

3.3.2 command_processor

Acting as the command dispatcher, this routine gets an input token (via getoken) and scans the command table for a match with a command name. When no token is present and the net is open a shell is forked off and telnet waits for it to die. Otherwise, module getcomm is called to scan the command table for a match between the token and a known keyword. If the token is ambiguous no action is taken. If no match was found in the command table, module execute is called to hand the line to a shell for execution as an unknown command[12].

3.3.3 getcomm

Module getcomm serves as the primary interface to the command table. When given a token it scans the entire command table[13] for a match with the token. Module compar is used to

---

[12] Note that any Unix command may be typed directly to telnet and executed in this manner. If a restricted version of telnet is desired, this is the place to modify the code. An error message may be generated instead of attempting to execute the command, and no shell would be executed on a tokenless line.

[13] The scan is done linearly, since the command table

evaluate whether or not the token matches a command table entry. Note that exact matches take precedence over partial matches in order to resolve the ambiguity between command pairs like 'ec' and 'echo'. If, during the scan, a match of n characters is found when there was a previous match of n characters, the token is considered ambiguous and a dummy command table pointer is returned (address of the last entry in the command table), while zero is returned for no matches found.

3.3.4 getoken

This module parses a token from a character string and returns a pointer to its start. If the passed pointer is zero, the starting point of the scan is found from the global 'nextoken' which is also set in this routine. Blanks and tabs are scanned off, then scanning continues until a break character (a char in the external array 'breaks') is encountered. If there was a token present, it is null terminated and a pointer to its start returned. Otherwise, a zero is returned.

3.3.5 getinpc

As the name implies, this routine gets the next input character and returns it to caller. Terminal buffering is done in the instruct structure, which is similar in theory to the standard Unix version six getc structure. The matter is complicated by the addition of the character mapping facility (allowing translation of a given character into an arbitrary string). If the character at the mapping pointer (instruct.def_expan_str) is non-zero, it is a signal that we're in the middle of mapping a character, and the next char in the map string is returned (and the pointer incremented). See also map_echo, below. Otherwise, we return a character that was read from the terminal and buffered in the structure (in instruct.data[] to be precise) unless there are none left, in which case a read from the terminal is done. In order to have the Unix newline character come out as cr-lf for telnet, if the character to be returned is a newline it is made into a carriage return[14] if telnet is in linemode[15].

---

is not outstandingly large. A binary chop may be used if many more commands are added, with minor modification to the compar routine.

[14] Note that this is not done for the mapped strings. They are considered to be literal.

[15] This should not be only linemode, because the Unix newline translations also depend on the setting of the stty nl translation bit.

3.3.6 buffer_full_line

This routine simply gathers up characters in the external character array netbuf until an endline character is seen. Also provided here are the line editing provisions of erasing the entire partial line and erasing the last character saved. Further, if the character is a DELETE or EOT and no connection is open, telnet will exit. The number of characters saved in netbuf is returned.

3.3.7 map_echo

Here is the routine which does the remainder of the funny character translations. Accepting one char as a passed parameter, map_echo translates it into the appropriate character and returns the translation. If the character is the Unix escape char (backslash), the next character is taken literally, and the backslash is swallowed. Otherwise, the character is used to index into the external character array maptab, which yields either the character itself or a value greater than 0177. If the 0200 bit is on, the character has a mapping on it, with a pointer to the replacement string given by string_tab[define_tab[char&0177]]. If there is not other mapping in progress (recursive mappings are not allowed), the define pointer in the instruct structure (instruct.def_expan_str) is set to the start of the replacement string in string_tab. The character selected is then written on the standard output if Unix is not echoing it, telnet is supposed to echo, and the echo has not been suppressed thru the external (char) bitmap echomask. Finally, it is returned to the caller.

3.3.8 netopen

Netopen opens a network connection. It has no parameters and gets any information needed from global data (such as netbuf). Since multiple concurrent connections are not allowed in this version, netopen will return if called with a connection already open. It first parses the hostname, if present, out of netbuf via getoken and constructs the special file name necessary[16]. Next, it calls module loadopnparams which parses off the rest of the command line and stores the appropriate data in the (global) openparams srtucture. In the event that

---

[16] If no hostname is present, 'anyhost' is used and the host number is expected to be in the openparams structure when the open is done. This and other parameters are parsed and loaded by module loadopnparams.

loadopnparms returns an error indication, the attempt to connect is aborted. Otherwise, if the host has been specified (either by name or number), the open on the net is done, after preparations are made for the user to abort it if desired. In order to assure that a host is really dead, the connection is attempted twice - the NCP is supposed to send a reset to the host if the first attempt fails. If the open fails again some hopefully informative diagnostic is typed on the terminal, using the value of the ubiquitous external errno that the system provides, and the routine returns an error indicator. Otherwise, initial option negotiation is carried out, the companion process forked off, and the terminal mode is set as specified by the external savemode[17] in a switch statement.

3.3.9 loadopnparams

Here we parse off the extra parameters to the connect command and load them into the (global) openparams structure. The structure is zeroed to remove any old data and any telnet default values (such as the timeout of 120 seconds) are initialized. Starting at the passed character pointer, the string is scanned for parameters which are loaded into the structure as they are encountered; the scan terminates when there are no tokens remaining in the parameter string. If an unrecognized parameter is encountered, the scan is aborted and the routine returns to the caller with an error indication.

3.3.10 netclose

Taking no parameters, netclose will close the net connection, mark it thus, and reset everything to normal. Included in that catch-all statement are simple functions such as resetting the terminal to the mode it had on entry, killing the companion process, and waiting for it to come home (via wait system call). In netclose, the interrupt signal is diverted to a routine called nullfunc which does nothing; the purpose in this is simply to allow a user to abort the waiting for the child to die.

3.3.11 option

This routine is used to carry on the telnet option negotiations. A caveat is immediately needed: due to the lack of reli-

---

[17] The value of savemode is set by the set_*mode routines (such as set_lienmode, etc.) and has no effect until a connection is opened. That is, the actual mode is not altered until a connection is opened, but remembered in savemode.

able communication between the parent and child processes, the range of the options negotiation that may be implemented reasonably is very limited[18]. At most, the intentions of module option can be described. Briefly, there are four static bit maps which describe the option status; the maps are kept in ints, which restricts the number of supportable options to sixteen, but longs would work just as well. These status flags, and what they represent are:

opt_in_effect	bits set for each option in effect currently
opt_requested	marks those that have been requested of the other side, but not yet replied to
opt_implemented	marks those options that telnet will consider and can support
opt_desired	is a place to remember settings requested while the connection was closed that should be negotiated upon opening a connection.

In addition, there are three additional maps with which to keep track of the status of the foreign host which have the same names with an 'f' prepended (opt_implemented is omitted) which are currently not used.

The routine consists mainly of a switch statement to decide what to do with the requested action passed as a parameter. In the interests of modularity, one of the actions is OPTINQY which will return an indication as to the current status of the passed option. The static bit maps are not made external/global in an attempt to make telnet a bit more structured. Similarly, the OPTINIT case does the initial negotiation for all the options the user requested prior to opening a connection. It scans the map and outputs telnet will and do protocols for any appropriate option. There is also a case for each of the telnet protocol messages will, wont, do and dont. They formulate a response and send it to the net.

---

[18] Actually, there are several schemes for doing option negotiation, but all are rather cumbersome. The main trouble is the fact that the standard Unix signal mechanism is hazardous to the life of system calls. Please refer to the later sections on IPC, options, etc. for further discussion.

3.3.12 set_cmd

Module set_cmd is the dispatcher for all commands which are prefixed by the set command. It is called thru the command table with two parameters: a word of data and a word of flags. If data is zero, it was called due to a set command and must parse off the next token to deduce it's action. If the next token is known, data and flags are set from it's entries in the command table; if not, they are set to cause an error. The flag word is then tested against the FPSET mask to determine if the command may be prefixed by the set command, and if not, a message is generated and error indicator returned. Thus assured that the data is valid, we enter a switch to cause the appropriate routine to be called for action.

3.3.13 send_cmd

Module send_cmd is similar to set_cmd, but has additional capabilities. First, if a net connection is not open, it returns an error indication (no place to send). If the data word is nonzero, and greater than 0177, it is assumed to be a protocol command (such as 'synch' or 'ayt') and the byte is passed to send_proto for transmission. If the word is less than 0200, it is sent just as is. On the other hand, if the data was zero, this is indeed a send command. The send command parameters are parsed off the command line via getoken and interpreted as long as tokens remain. If any token is not of correct form[19], an error message is generated and it is skipped. The desired byte(s) are then written to the net.

3.4 Minor Modules

The term 'minor modules' describes those routines which have little or no impact on the flow of control of user telnet. These are used by the major modules and each other (in some cases) to prevent obscuration of the problem with messy details. Again, they are not presented in their order of appearance in the code itself and have been shuffled in the hopes of painting a semi-coherant picture.

---

[19] The forms allowed are in the specifications book, but briefly they are:

nnn	to send a numerically specified byte. Interpreted as octal if the first digit is a zero.
/c	to send the control character corresponding to character The high order three bits are zeroed.
cmd	a command which may be prefixed by the send command

3.4.1 set_echo

Called with two parameters, set_echo sets the echomode flag and, if necessary, calls chngtty to effect the stty system call. The parameters are produced by the command table and consist of the setting desired and the usage flags. In the case of setting SETECHO, the routine was called by module set_cmd, and the desired setting is parsed from the command line and echo set according to it. Although the option negotiation routine is not currently notified of the change, the proper call is in the code, commented out[20].

3.4.2 set_endline

Having no parameters, this routine sets telnet's idea of the line termination character to the first character in the next token on the command line. If no token is present, no set is done. As usual, the token is extracted via getoken. The endline character is also added to the list of break characters (external character array breaks[]) that getoken uses in parsing. Finally, the current setting of the endline character is typed on the terminal - this means that if no token was present, the routine displays the endline character instead of setting it.

3.4.3 set_escchar

Set_escchar is almost identical to set_endline; the difference being the place the escape character is stored. The external (character array) prmpt[] contains the telnet command prompt, which is made up of some padding with the current escape character in the middle. Thus, the only place hit by set_escchar is prmpt[1], and all references to the escape character are done on that location. The prompt/escape storage was done in this manner to relieve any ambiguity that may have resulted from running telnets withing telnets - this way, if you've set distinct escape characters, you always know which telnet you're sending a command to.

---

[20] It seems reasonable that the following settings should be recognized:

```
on          >Immediate control of Local echo only.
off   /

local     >Begin option negotiation to set who echos.
remote    /
```

3.4.4 set_tty

Although the name implies that this routine is the workhorse of the terminal mode setting business, it has nothing to do with that. Initially written as a debugging aid, and included as a means of finding out where you are, this routine types the telnet terminal mode flags and forks off a child[21] to execute stty (I) for the system's idea of the terminal status.

3.4.5 set_charmode

This routine sets telnet into character-at-a-time mode. All that it does is reset from line mode (calls reset_linemode) and sets the appropriate flags.

3.4.6 set_msgmode

Analogous to char mode, we reset from line mode and set flags appropriately.

3.4.7 set_tenexmode

More of the same type of stuff. This routine simply calls set_charmode and set_echo, and then sets the proper flag.

3.4.8 reset_linemode

In addition to flag setting, this routine sends a signal to the companion process[22] to indicate the transition in mode and then invokes module chngtty to set the terminal into raw mode, etc.

3.4.9 set_linemode

Inverse of reset_linemode: sets the flags, and sends a signal to the child (companion process) to indicate the mode change. The signal is not sent if the companion is just starting, similar to reset_linemode.

---

[21] This is NOT done by module execute because it diddles the terminal modes before and after it forks. The true state of the terminal would thus not be revealed unless settty did a fork on it's own.

[22] The signal is NOT sent when the companion has just been started up. This is determined by the flag parameter and done to avoid killing the child before it can get set to handle signals correctly.

3.4.10 chngtty

Used to set and reset the terminal mode bits, this routine takes two integer arguments representing the mode bits to set and reset, respectively. It then calls stty2 with the mode word created from the expected boolean functions of the current mode word and the set/reset masks.

3.4.11 stty2

This routine sets the mode word of the terminal to its integer parameter. To avoid doing unnecessary stty calls, this routine only calls stty if the desired mode is not the same as the current mode[23].

3.4.12 pushtty

Working in conjunction with poptty, this routine provides a small stack for remembering terminal mode bits during temporary states (like opening of net connections and executing Unix commands). The stack resides in the external array ttystack, and the top is in the external ttystkp.

3.4.13 poptty

Inverse of module pushtty.

3.4.14 send_proto

Two-byte telnet protocol requests go thru this routine. Taking the byte of protocol as a parameter, it prepends the telnet IAC byte and sends the message off down the net. If the protocol request is a DM (data mark), the companion is told to expect an INS from the other side of the connection (via a signal to it) and the sendins routine is called to accomplish what the name says.

3.4.15 helper

Acting as a switch for the various help functions, this module takes care of invoking them correctly. Currently, it provides for the help command (by parsing the command line for it's args, and then executing the help program via execute), for the hosts command (by passing the buck to the hosts routine), for the

---

[23] Current mode is found in the curtty array, which represents the stty buffer format in a 3 word array. This makes the modes word curtty[2], and hence the name stty2.

commands command (by listing the keywords from the command table), and for the status command (by using execute to run the netstat program). The various cases are distinguished by the constants in the command table, which the command processor uses to call this routine.

3.4.16 execute

Performs the gyrations to run another program from telnet. This is not as bad as it sounds - just push the terminal mode and set to the original (entry) mode, do a fork (and exec if the child, wait if the parent), and pop the terminal modes to reset them. Frills and complications include having the parent ignore the quit and interrupt signals while waiting, and typing a prompt to indicate the end of the command's execution when the wait call completes.

3.4.17 bye

Generally a cleanup routine in case something ghastly happens. It closes the net connection (via netclose), resets the terminal to the mode it had on entry to telnet, and calls exit.

3.4.18 hosts

This module fields requests for host status, at a lower level than helper. It prepares to run the hosts program, and then has execute do it. If the parameter is non-zero, it is assumed to be a pointer to a string which is appended to the command line - that's how status info on specific hosts can be obtained.

3.4.19 load_char_set

This routine is a part of the (partially implemented) character mapping facility (see also mkcharmap (I) an appendix for information). It tries to read a previously constructed character map set into the mapping tables et. al. in core. The two parameters represent a string pointer and usage flag, respectively. If the pointer is null, the map set file name is assumed to be the default of 'character_set', otherwise, the name is parsed off. If the name does not begin with a '/' character at this point, the user's default working directory (obtained via gethomedir) is prepended to it. The file is then opened and read directly into the tables. If the user specifically requested the load (e.g. this routine was called thru the command table) a success/failure message is typed to the terminal. This way, explicit loads of default maps may be coded into the entry section of the code, although this has not been done yet.

3.4.20 delay

A means to waste some time, delay invokes the sleep system call with the number of seconds as specified in it's parameter.

3.4.21 prompt

Writes the command prompt on the terminal if a net connection is not open.

3.4.22 compar

Used to determine character string matching for command parsing. Given pointers to two strings, compar will count how well they match and return zero for an exact match, and an integer indicating the number of characters matched otherwise. If the mismatch occurs on the null terminating the first string, the number of characters +1 matched is returned; otherwise, this number negated is returned.

3.4.23 strmove

Copies the string pointed to by the first parameter to the destination address specified by the second parameter. The copy ends after a null is reached. Returned is the address of the null terminating the destination string, so the routine may be called repeatedly to append strings with a minimum of fuss.

3.4.24 setmember

Determines, predicate style, if the character specified by the first parameter is in the string whose address is given by the second parameter. In other words, if the first parameter (character) is a member of the set represented by the second parameter (pointer to a character string), a non-zero (true) is returned. A zero (false) is returned if the character is not in the set.

3.4.25 number

Converts strings of ascii digits into integer/binary format. If the number begins with the character '0' it is assumed to be octal, otherwise decimal. Any non-digit (or non-octal digit if the number is supposed to be octal) terminates the scan. Additionally, if a non-octal digit is found in a supposed octal number an error message is typed.

3.4.26 bit_on

Also a predicate. True if the bit number specifed by the first parameter is set in the bit string specified by the second

parameter (which is actually a pointer to an array of bytes); false otherwise.

3.4.27 set_bit

Sets the bit specified by number in the first parameter in the bit string specified by the second parameter. See bit_on for some details.

3.4.28 reset_bit

Inverse of set_bit.

3.4.29 gethomedir

Extracts the home directory of the current user from the password file (the password itself is not used/needed). Returns a pointer to the string containing it if successful, zero if unsuccessful.

3.4.30 colon

Given a pointer to a string, scans for the next colon and returns a pointer to the character after it. Used by gethomedir to parse the password file.

3.5 About the Companion Process, Usrtelnetin

In order to achieve a full-duplex connection under Network Unix, one process is required for each direction. User telnet provides communication between the user and the network, the companion provides communication between the network and the teleprinter (screen, etc.). As the companion process need not cope with the user's requests (and cannot cope with the option negotiation, as we shall see later) it is rather simple.

3.5.1 General Overview of Functioning

The main loop of the companion consists only of a trivial 'forever' loop. For each traversal of this loop, one character is input from the network and acted upon. If the character is outside of the Telnet Ascii range (greater than 127.), it is interpreted as a protocol command[24]. The character is then

---

[24] The obvious drawback to this scheme is the prohibition of Telnet's binary transmission option. However, it remains as a measure of grace for the old telnet protocol.

deleted from the buffer, and the process repeated. Of course, the character is output to the terminal if it is in the normal Ascii range.

3.5.2 Telnet/Companion Communication

There are several reasons telnet may want to direct the actions of the companion; these include state changes at the user's request and extra-ordinary conditions that may arise from them. Provided only with the Unix standard Inter-Process Communication (IPC) facility of signals, telnet must make do. A brief explanation of the signals used, and their functions follows. Signals will be denoted by their mnemonic names (which appear in the parameter file param.h).

3.5.2.1 SIGINT and SIGQIT

These signals are usually generated directly by the user from keys which have a special 'out of band' meaning to Unix: process interruption and interruption forcing a core dump, respectively. Their semantics restrict their use to conditions where the user has requested that telnet abort operation. Because the companion is viewed as a slave of user telnet proper, they are ignored.

3.5.2.2 SIGHUP

This signal denotes the disappearance of the carrier on a modem connection under normal usage. In the current context it is viewed as a condition which should cause termination of the connection, a generalization of the original usage. When user telnet wishes the companion to die gracefully it sends this signal, which the companion then traps (in routine huptrap). Because this is the normal exit procedure for the companion, it prints a brief message to the user's terminal informing him/her of the loss of the connection. Note that if the user terminal is indeed a dialup which loses carrier the expected exit occurs.

3.5.2.3 SIGINR

This signal is not present in standard Unix, and was added to support the network's capability for out of band signals on the control link. The telnet protocol uses this feature to implement its synch sequence. Consequently, receipt of this signal causes counters and flags to be set to indicate the expected appearance of a telnet data mark in the input stream.

3.5.2.4 SIGPIPE

This signal has no relevance to telnet and has been pirated for another purpose entirely. When the user requests a synch se-

quence sent telnet informs the companion to expect an 'interrupt' from the net. The trapping routine (expect_an_ins) sets flags to indicate that this signal was received so the input routine does not get confused about the error return from its read of the network.

3.5.2.5 SIGSPA

This signal is undefined in standard Unix, but allowed for, and is named as the first 'spare' signal. The routine that traps this signal toggles a flag which causes carriage returns to be ignored or passed to the user's terminal, dependent on the mode of the local terminal.

3.6 Comments in Hindsight

User telnet has several problems that are not easily corrected. Most of these small griefs are direct results of the initial design of program structure[25] and subsequent patches to relieve the problem. Refer to the appendix regarding the need for IPC for more information on this class of problem. Some of the troubles are just a result of patch after patch over an extened period of time to increase the features and/or performance of user telnet, the code has become increasingly cumbersome and obtuse. In general, this code for user telnet provides a reasonable conceptual base for the complete rewrite which will be needed to make use of enhanced Unix IPC.

3.6.1 Bugs and Other Vermin

There are a few rough spots in user telnet that could use some attention, but they are minor annoyances and should cause no trouble to the casual user. Without delay then, a list:

Option negotiation cannot be reasonably implemented as both telnet processes need to be aware of the state changes of the other, or request changes of state based on commands received from the network or user keyboard. See the appendix on telnet with IPC for possible solutions.

While on the topic of options and protocol, it should be pointed out that this version of telnet does not handle

---

[25] When user telnet was first designed the effects of Unix I/O calls blocking was discounted. This lead to the split into two processes to reasonably handled the duplex nature of a telnet connection, which lead to the problems of each simplex process knowing what the other was doing.

the SGA option correctly for Platform.

Some improvement could be made in the handling of the backspace and line delete characters, especially in character mode. Currently, the local characters are not mapped into the telnet command sequences which forces the user to be familiar with the conventions used on the host (s)he is connected to in this regard. This 'feature' allows the remote host to appear local to the last detail while in character mode and tenexmode, but not while in linemode or msgmode. Perhaps what is needed is the creation of a new mode which is a cross between charmode and msgmode, thus: called charmode, it acts like the current charmode but does certain translations (like erase characters) automatically. The mode that used to be called charmode could be renamed to (something like) imagemode, leaving the way open for a full binary transmission mode (binmode?).

As long as the topic of connection modes is on hand, a suggestion is in order. There is a separate routine to switch to each mode available; while this is good programming practice, it forces switch statements to be repeated in several places in the code. Perhaps one routine (set_tmode) to handle all transitions would make the code a bit less verbose.

3.6.2 Possible Enhancements

Several additions could be made to user telnet to increase its usefulness and flexibility. Among these are the addition of a switch to set the terminal mode from the invocation line, and/or the use of a TOPS-10 style switch initialization file in the user's home directory. These class as conveniences, like the tenexmode, and may be safely ignored. The outstanding enhancements to user telnet are unachievable with the current Unix system due to the lack of reasonable IPC.

For example, option negotiation is not carried out in anything like a straightforward manner. In fact, once a network connection is opened and the companion process spawned, telnet proper does not read from the network at all. Thus the companion must deal with requests for option negotiation in the only way possible - stark refusal to change state. Of course, as with all the world's ills, this too can be remedied with IPC facilities. Refer again to the IPC appendix.

3.7 Installation of User Telnet

The steps required to get user telnet ready to run are outlined in this section. Primarily, an explanation of the actions of the 'Run' shell file kept in the telnet home directory is provided. As an extra added feature, a list of the files needed to accomplish the 'Run' file's execution is also provided. All references to files that are not full pathnames are taken from the telnet home directory.

3.7.1 List of Files for User Telnet

This is intended to serve as a guide, if not a definitive list, of the files needed to maintain and run user telnet. Of course, for exact treatment of how the files and programs are used and configured, the Run shell file is the best resource for you.

Names for telnet:
  /usr/bin/telnet
  /usr/bin/con

User telnet home directory (at Illinois):
  /sys/netsys/ncpp/utel

Current version(s):

user telnet	4.0 (I1.37)	(usrtelnet.c)
companion	4.0 (I1.2)	(usrtelnetin.c)

Parameter files (specified from the user telnet home directory):[26]
  ../../h/net/usrtelnet.h
  ../../h/net/telnet.h
  ../../h/param.h
  ../../h/user.h
  ../../h/net/mkcharmap.h
  ../../h/net/netopen.h

Accessory routines (called for some of the commands) and the files they access :

/etc/hosts	(hosts.c see next section)
/etc/help	(aux/help.c)
/mnt/help/telnet	
/etc/netstat	(netstat.c see next section)

---

[26] include files to be in /usr/include. Check your local documentation and conventions for exact loction.

(several that won't be enumerated here,
principly the files in /dev/net)

N. B. The choice of the program to perform this function is dependent on which IMP inter- face your system has.

3.7.2 The Run Shellfile

This file may be used to bring up user telnet. Consult the listing of the file. Note that it takes an optional argument; this is included for the purpose of dress rehearsals. In other words, the command 'Run /tmp' will execute the Run file, and act as if /tmp were the root of the file system. Directories like /usr/bin and /etc will need to be present in the fake root in order for the command file to work correctly.

```
: this Run shell file compiles all programs needed by user telnet.
: the executable files, and any text files are moved to the
: appropriate directories.
: This shell file may be tested by executing it with an argument.
: This argument is taken to be the -root- of the filesystem, and
: all file creations are done with respect to it.  see the code.
:
: further debugging aid is provided by using the optional SECOND
:  argument, which will cause an experimental version to be
:  generated with slightly changed names if it is -x- or -X- .
:
: Files are set to be owned by user bin of group bin.
:
:
: NOTE : since the telnet version number is part of the
:        name of the source file, this file must be editted
:        when release of a new version is desired.
:
:
: must have null or valid args
if $1x = x              goto chk2
: else, need to see if first arg writable
  if -w $1            goto chk2
: else, bad first arg, not writable
  echo cant use $1 as the fs root.
  exit
: chk2
: make sure the second arg is -x- or -X- if its there
if $2q = q  -o  $2q = Xq  -o  $2q = xq              goto plowon
  echo invalid second arg -$2-
  exit -1
: plowon
: finally ready to start off with the user telnet program itself
if $2q != q          cc -O -s -n -o telnet -D xtni usrtelnet.c -lj
```

```
if $2q = q          cc -O -s -n -o telnet usrtelnet.c -lj
: done compiling, switch off for the renaming...
if $2q != q         goto linkxver
  : make sure that the links dont get in the way.
  :  -f switch to rm says zap file even if not 0222 mode
  rm -f $1/usr/bin/con $1/usr/bin/telnet
  mv telnet $1/usr/bin
  chmod 555 $1/usr/bin/telnet
  chown bin $1/usr/bin/telnet
  chgrp bin $1/usr/bin/telnet
  ln $1/usr/bin/telnet $1/usr/bin/con
  goto usrteldone
: linkxver
  : experimental - change names to protect innocent (user)
  rm -f $1/usr/bin/xcon $1/usr/bin/xtelnet
  mv xtelnet $1/usr/bin
  chmod 555 $1/usr/bin/xtelnet
  chown bin $1/usr/bin/xtelnet
  chgrp bin $1/usr/bin/xtelnet
  ln $1/usr/bin/xtelnet $1/usr/bin/xcon
: usrteldone

: the auxiliary programs and text files are in a sub-dir.
chdir aux
: now proceed to the companion process.
if $2q != q         goto xcomp
  : compile and set up the normal production version.
  cc -O -s -n -o usrtelnetin usrtelnetin.c -lj
  rm -f $1/etc/usrtelnetin
  mv usrtelnetin $1/etc
  chmod 555 $1/etc/usrtelnetin
  chown bin $1/etc/usrtelnetin
  chgrp bin $1/etc/usrtelnetin
  goto compdone
: xcomp
  cc -O -s -n -o xusrtelnetin  usrtelnetin1.c -lj
  rm -f $1/etc/xusrtelnetin
  mv xusrtelnetin $1/etc
  chmod 555 $1/etc/xusrtelnetin
  chown bin $1/etc/xusrtelnetin
  chgrp bin $1/etc/xusrtelnetin
: compdone

: help functions program
: N.B. this file is a link to the ftp help program....
cc -O -s -n help.c -lj
rm -f $1/etc/help
mv help $1/etc/help
chmod 555 $1/etc/help
chown bin $1/etc/help
chgrp bin $1/etc/help
```

```
: char map creating program
cc -O -s -n mkcharmap.c -lj
rm -f $1/usr/bin/mkcharmap
mv mkcharmap $1/usr/bin
chmod 555 $1/usr/bin/mkcharmap
chown bin $1/usr/bin/mkcharmap
chgrp bin $1/usr/bin/mkcharmap

: ok, done there, now have documentation to distribute.
chdir ../doc
: the help text
cp usrtelnethelp $1/mnt/help/telnet
: actually, ehis could be an nroff command of the helpgen file plus
:  the style macro package, but that would be several more absolute
:  pathnames, which isnt a good idea to do.  see the doc directory
:  under network documentation.
chmod 444 $1/mnt/help/telnet
chown bin $1/mnt/help/telnet
chgrp bin $1/mnt/help/telnet

: and the man page
: which really shouldnt be here at all, but in the documentation d
cp telnet.1 $1/usr/lpd/manuals/man1
chmod 444 $1/usr/lpd/manuals/man1/telnet.1
chown bin $1/usr/lpd/manuals/man1/telnet.1
chgrp bin $1/usr/lpd/manuals/man1/telnet.1

: and mkcharset man page.
cp mkcharmap.1 $1/usr/lpd/manuals/man1
chmod 444 $1/usr/lpd/manuals/man1/mkcharmap.1
chown bin $1/usr/lpd/manuals/man1/mkcharmap.1
chgrp bin $1/usr/lpd/manuals/man1/mkcharmap.1

: not quite finished yet...  need to do the misc progs.
chdir ../../hosts
: now for the fasthosts program
cc -Osno fasthosts fasthosts3.c -lj
: orthogonal program, if already there, leave it alone.
mv fasthosts $1/etc
chmod 555 $1/etc/fasthosts
chown bin $1/etc/fasthosts
chgrp bin $1/etc/fasthosts

: and the IMP/interface status program.
: since interface may change,
:  you may not want this version.
chdir ../netstat
: version for the ACC interface.
cc -O -s -n netstat.c -lj
: orthogonal program, it already there, leave it alone.
mv netstat $1/etc
```

```
chmod 555 $1/etc/netstat
chown bin $1/etc/netstat
chgrp bin $1/etc/netstat

: finally, done.
```

4 Appendices

The materials in these appendices are largely replications of other documents. The manual pages found here are, for example, snapshots of the usual Unix manual pages that may be accessed via the man command (man (I)). These should be up to date, which is not guaranteed for the snapshots in the appendix.

4.1 Manual Pages

Only those pages that would be useful to the telnet user are included here. For other pages, use man (I) to produce copies.

4.1.1 User Telnet

The User Telnet manual pages are reproduced on following pages.

4.1.2 Mkcharmap

The mkcharmap manual page is reproduced on following pages.

NAME

telnet - Converse with a host via the network

SYNOPSIS

telnet [hostname] [<parameters>]
con [hostname] [<parameters>]

DESCRIPTION

Telnet allows a user to communicate with a remote time sharing system across the Network (this document reflects version 3.9 (11.36) of the user telnet code). Parameters to telnet are optional; if none are given telnet will enter a command accepting mode. If present they may consist of a host name followed by switches of the form `-k'. Note that the Unix command `telnet' and the telnet command `telnet' (and its synonyms) are syntactically identical; however, to open a connection, a host must be specified.

Most switches take additional argument(s) which are usually integer(s). Telnet will interpret any integer as decimal, unless the first digit is a `0' - in which case it will be interpreted as octal. Switches and their arguments (with integers represented as `#') are:

-a # desired message allocation size

-d # # specifies a direct connect (as opposed to normal ICP) socket. a 32 bit quantity which is specified as two 16 bit integers, high order portion first. usually, one wants the s switch, not this one.

-h # host number of the desired host (use without the hostname)

-s # foreign socket to connect to (default is 1)

-t # number of seconds before timeout (default 120)

The following table is intended to be a fast reference guide for telnet users. It lists all keywords user telnet will recognize, their prefixes, arguments, and a very brief description of the command. For an introduction to user telnet and/or more detailed information on the commands, refer to the Telnet User Manual (to be found in the documentation directory), or use the help command from telnet. Items enclosed in square brackets ("[", "]") are optional; commands enclosed in double braces ("{{", "}}") are not yet implemented; telnet protocol mnemonics are enclosed in single quotes ("`", "'"); arguments are expressed by mnemonics enclosed in angle brackets ("<", ">").

PREFIX	KEYWORD	PARAM(S)	DESCRIPTION
[send]	ao		'Abort Output' to server
[send]	ayt		'Are You There' to server
[send]	brk		'BReaK' to server
	bye		Synonym for end
[set]	charmode		Character-wise I/O
	close		Terminate connection
	commands		List command keywords
	connect	[<parameters>]	See DESCRIPTION
[send]	del		DELete (0177) to server
	disconnect		Synonym for close
{{[set]	display	<c> "string"	Map a char from net}}
send	ec		'Erase Char' to server
[set]	echo	<mode>	Local echo "on" or "off"
send	el		'Erase Line' to server
[set]	eline	[<c>]	Change or see endline char
	end		Exit telnet (closes conn.)
[set]	escape	<c>	Change or see escape char
	exit		Synonym for end
[send]	goa		'GO-Ahead' to server
	help	[<keyword>]	Help for keyword spec'd
	hosts		List network hostnames
[send]	ip		'Interrupt Process'-server
{{[set]	key	<c> "string"	Map a char from keyboard}}
[set]	linemode		Line-wise I/O
	load	[<pathname>]	Load pre-made char maps
[set]	msgmode		Message-wise I/O
[send]	nop		'No-OPeration' to server
[echo]	off		Set local echo off
[echo]	on		Set local echo on
	open		Synonym for connect
{{	option	<opt> <mode>	Negotiate an option}}
	quit		Synonym for end
	send	<cmd> <args>	Command prefix for novices
	set	<cmd> <args>	Command prefix for novices
	settty		Inspect modes (for debug)
	status	[hostname]	Get host or IMP status
[send]	synch		'Synch sequence' to server
	telnet	[<parameters>]	Synonym for connect
[set]	tenex		Charmode with echo off
	wait	<nsecs>	Suspend keyboard input

FILES

/dev/net/<hostname>	special file for each host
/dev/net/anyhost	special file for anonymous host
/etc/passwd	for home directory lookup
/etc/help	help lookup program
<help>/telnet.hlp	help text file
/etc/usrtelnetin	companion process program
/etc/fasthosts	host status program
/usr/bin/netstat	IMP/net-interface status program
/bin/sh	shell (to execute unknown commands)

SEE ALSO

mkcharmap(I), Telnet User Manual.

DIAGNOSTICS

Many. Nothing too weird though.

BUGS

Option negotiation is not done properly (telnet should issue an option request and wait for the server to issue a response - no waiting is done). No reasonable fix to this problem is possible without enhancing Unix IPC facilities.

NAME

Mkcharmap - Create a character map set for user Telnet.

SYNOPSIS

mkcharmap

DESCRIPTION

Mkcharmap produces something called a 'character map set file' for use by the user telnet program. This facility allows a user to define a set of keys which will not be interpreted literally by user telnet, but instead, will be mapped into an arbitrary (user-defined) character string. the mkcharmap program guides a user thru the generation of a map set and outputs it, on command, to a disk file (<home directory>/character_set) in the proper format for user telnet.

The commands to mkcharmap are:

bye — Exit the mkcharmap program

define — Define a character to any string of arbitrary characters and ask if it is to be echoed to the terminal when invoked. The command is self documenting. Backslash may be used to escape any character.

end — Exit the mkcharmap program

help — Type a little help on the terminal.

load — Load a character map set from '<home dir>/character_set'. Done automatically at initial execution if a 'character_set' file already exists.

print — Print the current set of character mappings.

save — Save the current character mappings in a file named '<home dir>/character_set'.

SCENARIO

This is an example of the use of the define command. It maps the control b character into a connection to bbn, execution of the 'tenex' command, and goes thru the bbn login sequence. It is important to remember that each line of the define string is terminated with a carriage return and an new line, so be careful when defining characters.

```
define
Character to be mapped:^B
Enter String (end with cntrl c)
con bbn<cr><lf>
^wait 10<cr><lf>
^tenex<cr><lf>
```

```
^echo off<cr><lf>
log holmgren pass acct <cr><lf>
^echo on<cr><lf>
^C
Do you want this string echoed(y or n):n
```

FILES

/etc/passwd	for home directory lookup
/etc/mkcharsethelp	help text file
/bin/sh	shell (to execute unknown commands)

SEE ALSO

telnet(I), Telnet User's Manual.

DIAGNOSTICS

Self-Explanatory

BUGS

The only character map set file that can be accessed directly is <home directory>/character_set.
Not all non-printing characters are escaped in the printing of a map set.

4.2 On the Question of User Telnet and IPC

There seems little doubt that something needs to be done to allow Unix telnet to (at least) do option negotiation in a reasonable manner. The basic problem revolves about the fact that a Unix process is suspended (blocked) when it requests I/O that cannot occur immediately. As stated in several papers produced under this contract[27], some IPC mechanism is needed to either reliably synchronize the two telnet processes, or inform telnet when it is possible to do I/O without being blocked. Since the network connection is a full duplex affair, any one of the four I/O operations (read/write terminal/network) possible can block telnet, making it appear hung for the others; so each half of the connection is managed by one process. Thus the solving of the blocking problem produces a problem in altering mode - neither process can reasonably find out what the other has seen in the way of requests to change mode.

For example, the user types a command to switch to remote echo. The terminal to network user telnet process sends out the appropriate protocol to accomplish the change. When the foreign host's reply reaches the network to terminal process, it sees a new request to change state, as it didn't know about the user's command. Conversely, if the remote host desires a change, the keyboard to network process will never find out that protocol came in from the net requesting the change without reading the net (or a pipe back from the companion) and, just maybe, blocking on that read.

Assuming the existance of a reasonably flexible IPC facility, several of the sore spots in telnet can be repaired. Some redesign is required to adapt to the new facilities, but most of that work is focused on the interfacing and major flow of control. Much of the code in both the user and server telnet programs will survive the change-over. However, that topic is just outside the scope of this manual; refer to the Telnet Functional Description also produced under this contract.

---

[27] Refer to the Request for Comments on Unix InterProcess Communication, the Telnet Functional Description, and Telnet Program Specification produced under this contract for discussion of the problems and solutions presented.